Efficient Programming of Nanowire-based Sublithographic PLAs: A Multilevel Algorithm for Partitioning Graphs

Vivek Rajkumar (University of Washington CSE)
Contact: <vr@u.washington.edu>
California Institute of Technology, CBSSS program 2004

1 Introduction

Our CBSSS project dealt with the efficient programming of nanowire-based sublithographic programmable logic arrays. A mesh of nanowires with PLAs at particular junctions can be logically programmed to fit logic functions. However, doing this efficiently is a problem, since looking at all possible combinations is a lengthy process. However, the potential benefits are staggering: "Key features can be a few nanometers wide, a few silicon atoms wide, perhaps the ultimate scale for devices. This allows us to design computing components without the costs or limits of ultra-fine lithography". [1]

In addition, "under conservative assumptions using 10nm nanowires and 90nm lithographic support, [a yield of a] density around 500,000nm^2 [per] or term for a 60 or-term array [is achieved]; a complete 60-term, two-level PLA is roughly the same size as a single 4-LUT logic block in 22nm lithography". [2] Thus, we looked at different ways of solving the efficiency problem. For one, reducing a nanomesh into a graph is an option, since running graph-partitioning algorithms would allow us to partition the mesh such that particular partitions handled particular components of the logical statement. Then the overall graph can, in a sense, be "spliced together" to serve as the full function.

2 Algorithm

In this paper, I will describe a multilevel algorithm for partitioning graphs introduced by Bruce Hendrickson and Robert Leland, of the Sandia National Laboratories. The end goal is to apply the algorithm to benchmarks (given to us by Professor Andre Dehon [1, 2]) to compare it to other algorithms that attempt to efficiently program the meshes. I will be working with Kumar Jeev, Paolo Codenotti, and Akhsar Kharebov, who will each study max-clique partitioning, repeated min-cut partitioning and the architecture of nanomeshes, respectively.

---

[1] André Dehon, "Sub-lithographic Computing Systems" *appearing in* HotChips-15, August 17-19, 2003
[2] André Dehon, "Nanowire Based Sublithographic Programmable Logic Arrays" *appearing in* Proceedings of the International Symposium on Field-Programmable Gate Arrays, FPGA2004, February 22-24, 2004

As Hendrickson and Leland say, “The graph partitioning problem is that of dividing the vertices of a graph into sets of specified sizes such that few edges cross between sets. This NP-complete problem arises in many important scientific and engineering problems. Prominent examples include the decomposition of data structures for parallel computation, the placement of circuit elements and the ordering of sparse matrix computations.” [3] The algorithm in question works by breaking down a given graph into smaller and smaller graphs that are simpler to deal with and hence program. From this comes the “multilevel” attribute. H/L use a spectral partitioner for the smallest graph. As they say, the multilevel algorithm “produces high quality partitions at a low cost”. [4]

2.1 Problem Definition

The problem can hence be stated as (from H/L, although this statement is really the quest of all graph partitioning algorithms):

> Given a graph G = (V, K) where V = {v1, v2, ... v_n} and K = {e1, e2, ... e_n} where the edges are weighted, and the number of desired partitions P, how can one find a partition of V such that into P partitions such that the sums of the vertex weights in each partition are approximately equal, AND the sum of the weights of the edges that cross between sets is as small as possible? [5]

2.2 Hendrickson, Leland

Hendrickson and Leland's method works by coarsening the graph till it's small enough, then to partition the graph, and then while the graph is not equal to the original graph, uncoarsen it and its partitions [6]. As they say, “Coarsening is advantageous because is that the number of possible partitions grows exponentially with the number of vertices in the graph, so a good partition of the coarse graph is much easier to find than a good partition of the original one. The price paid for this reduction in complexity is that only a small number of the possible fine graph partitions are represented and are therefore examinable on the coarse graph”.[7]

---

[3] Bruce Hendrickson & Robert Leland, “A Multilevel Algorithm for Partitioning Graphs”, Sandia National Laboratories *appearing in* Association for Computing Machinery Inc 1995, p.1
[4] Hendrickson, Leland p.1
[5] Hendrickson, Leland p.2
[6] Hendrickson, Leland p.2
[7] Hendrickson, Leland p.2

2.3 Construction of coarse graph

H/L's construction of the coarse graph can be summarized as follows:

```
1. Find a maximal matching
2. For each matching edge {i, j}
     Contract edge to form new vertex v1
     Vertex_weight {v1} = weight {i} + weight {j}
     If i and j are both neighbors to a vertex k
        Edge_weight {v1, k} = Edge_weight
        {i, k} + Edge_weight {j, k}
     EndIf
   EndFor [8]
```

There are a number of benefits to using this method. H/L say, "This coarsening procedure has a number of attractive properties. First, any partition of the coarse graph corresponds naturally to a partition of the fine graph." [9] In addition, "since vertex weights are summed, constraints on the set sizes that depend just on the number of vertices in a set are preserved in a weighted sense in the coarse graph." [10]

Finally, "since edge weights are combined, any linear penalty function on the edges crossing between partitions will be preserved as a weighted metric under the coarsening process. Thus for most metrics of partition quality, a good partition of the coarse graph will correspond to a good partition of the fine graph." And of course "this coarsening algorithm is fast. The entire operation can be implemented in time proportional to the number of edges in the original graph". [11]

2.4 Partitioning of coarse graph

The coarsest graph is then partitioned, but the partitioning algorithm used here is relatively unimportant. H/L used a spectral method among others, but I just implemented a BFS since this graph was relatively small. After this, the partition is uncoarsened using this method:

```
1. For each vertex in smaller graph
     Union one or two vertices from larger graph
     Assign a vertex from larger graph to same set as coarse graph counterpart
   EndFor [12]
```

---

[8] Hendrickson, Leland p.3
[9] Hendrickson, Leland p.3
[10] Hendrickson, Leland p.3
[11] Hendrickson, Leland p.3
[12] Hendrickson, Leland p.4

2.5 Refinement of partitions

Now, an algorithm must be used for refining the partitions. The one H/L use is as follows:

```
1. While there exists a better partition
      Best_Partition = Current_Partition
      While Termination Criteria Unreached
         Select V to move
         Perform move
         Update gains of neighbors
         If Current_Partition is balanced and better than Best_Partition
            Best_Partition = Current_Partition
         EndIf
      endWhile
      Current_Partition = Best_Partition
   endWhile
```

There are a variety of termination criteria. The main condition is that there are no more possible moves from the larger graph to smaller graphs, among others. [13]

3 Results

The results that H/L observe are as follows, using the multi-level algorithm: [14]

3.1 Hammond Mesh

# Sets	Edges cut
2	119
4	236
8	393
16	652
32	1065
64	1688
(4,720 vertices, 13722 edges, 3.44 seconds)	

3.2 Barth5 Mesh

# Sets	Edges cut
2	196
4	412
8	648
16	1118
32	1779
64	2906
(15,606 vertices, 45878 edges, 6.17 seconds)	

[13] Hendrickson, Leland p.6
[14] Hendrickson, Leland p.12

3.3 Ocean Mesh

# Sets	Edges cut
2	499
4	1991
8	4743
16	9160
32	14628
64	22353
(143,437 vertices, 409,593 edges, 38.0 seconds)	

They observe that, "the local refinement scheme significantly improves the partitions generated by both the inertial and spectral partitioners and has modest cost. In fact application of KL may actually reduce net run time…because in the process of locally refining at one recursion level, subgraphs are produced [that] are more efficiently handled at the next level of recursion." [15]

4 Conclusions

The multilevel algorithm presented has been shown to be efficiently fast at partitioning several graphs of varying degrees of complexity. Because of this, once the nanomeshes have been reduced to a respective graph G, the multilevel algorithm can be applied to partition G efficiently, from where it will be much easier to program. As the partitioned nets will not be sparse, big chunks of the function can be programmed into these nets. Once each of the nets has been programmed, the overall graph (and hence nanomesh) will contain the logic necessary to evaluate the function. Future work will entail using this algorithm in combination with other partitioning algorithms, and studying the resulting efficiency.

5 References/Acknowledgements

I drew mainly from Hendrickson's and Leland's "A Multilevel Algorithms for Partitioning Graphs" from ACM 1995. In addition, the project was discussed and approved by Dr. André Dehon of Caltech[16], who has written a variety of papers on the topic of nanowire-based PLAs.

[15] Hendrickson, Leland p.12
[16] http://www.cs.caltech.edu/~andre/